SYSTEM AND METHOD FOR SPEEDING UP EJTAG BLOCK DATA TRANSFERS

Background

Field of the Invention

[1001] The present invention relates generally to integrated circuits and more particularly to efficient block data transfers over a serial test port.

Discussion of the Related Art

[1002] Complex integrated circuits have become increasingly difficult to test. One mechanism for testing these complex integrated circuits uses a Joint Test Action Group ("JTAG") test access port ("TAP") as defined in IEEE Standard 1149.1, *IEEE Standard Test Access Port and Boundary-Scan Architecture* (the "IEEE Standard"), which is incorporated herein by reference in its entirety for all purposes. The JTAG TAP provides an external interface to the integrated circuit whereby the integrated circuit can be debugged. According to the standard, a TAP is added to each integrated circuit. The TAP includes at least three inputs: a test clock ("TCLK"), a test mode select ("TMS"), and a test data in ("TDI") port. The TAP includes at least one output: a test data out ("TDO") port. Data is serially shifted from the TDI port and into the integrated circuit and serially shifted out of the integrated circuit and onto the TDO port. In this manner, "test vectors" may be written to or read from the integrated circuit via a test probe to determine whether the integrated circuit is operating properly.

[1003] Software debug on complex integrated circuits has likewise become increasingly difficult to conduct. In this regard, an extension of the JTAG standard, referred to as EJTAG, has been developed. EJTAG is a hardware/software subsystem that provides comprehensive debugging and performance tuning capabilities to processors and to system-on-chip components

having processor cores. EJTAG exploits the infrastructure provided by the JTAG Test Access Port (TAP) standard to provide an external interface, and extends an instruction set of the processor and privileged resource architectures to provide a standard software architecture for integrated system debugging.

[1004] Using EJTAG, instructions to be executed by the processor, in addition to data, may be downloaded to the processor via the test probe. Serially downloading instructions, which are executed as they are received by the processor, causes the processor to operate particularly slowly. One conventional mechanism to speed up this process is to download all the instructions to be executed by the processor to some portion of the processor's memory upon entering debug mode.

[1005] As serially shifting instructions and data to and from a processor through EJTAG is extremely time consuming relative to the operational speeds of the processor, existing EJTAG protocols (specifically, versions 2.5 and earlier) may require as many as 149 overhead test clock ("TCK") cycles for every 32 bits of instruction or data transferred. Accordingly, even at reasonable rates for TCK, transferring large blocks of instructions or data to and from the integrated circuit is often measured in terms of minutes.

[1006] What is needed therefore is a system and method for improving efficiency of block data transfer operations to and from a processor implementing EJTAG.

Brief Description of the Drawings

[1007] FIG. 1 illustrates a system for device testing and software debug on integrated circuits according to the present invention.

[1008] FIG. 2 illustrates various registers associated with a test access port according to the present invention.

[1009] FIG. 3 illustrates a transfer of data through a test access port in accordance with an operation of a conventional test system.

[1010] FIG. 4 illustrates a more efficient TDI-to-TDO path according to the present invention.

[1011] FIG. 5 illustrates a transfer of data through a test access port in accordance with an operation of the present invention.

Detailed Description

[1012] An embodiment of the invention is discussed in detail below. While specific implementations are discussed, it should be understood that this is done for illustration purposes only. More particularly, the present invention is described in terms of a particular Extended Joint Test Access Group ("EJTAG") implementation developed by MIPS Technologies, Inc., 1225 Charleston Road, Mountain View, CA 94043-1353 ("MIPS Technologies"), which is an extension to the JTAG standard mentioned above. A complete discussion of the EJTAG implementation is described in *EJTAG Specification*, Document Number MD00047, Revision 2.60, dated February 15, 2001, by MIPS Technologies (the "EJTAG Specification"), which is incorporated herein by reference in its entirety for all purposes. A person skilled in the relevant art will recognize that other components and configurations may be used without departing from the spirit and scope of the invention.

[1013] FIG. 1 illustrates a testing system 100 for testing an integrated circuit, which in this example is a target processor 110. Target processor 110 includes a test access port ("TAP") 115

that services a number of signals including Test Clock ("TCK"), Test Mode ("TMS"), Test Data In ("TDI") and Test Data Out ("TDO"). TCK and TMS control the state of a TAP controller (not shown), which is a state machine that controls access to certain registers within TAP 115. Access to these registers occurs serially through TDI and TDO. The structure and functionality of TAP 115 are described in greater detail below, and in the EJTAG Specification and IEEE Standard. Interfaced to target processor 110 is a target memory 120 resident either within target processor 110, external to target processor 110, or some combination of the two. Target processor 110 is any programmable semiconductor device including a microprocessor, microcontroller, System-On-a-Chip ("SOC") component (*e.g.*, ASIC, ASSP), etc.

[1014] Testing system 100 also includes a test probe 130 operating in conjunction with a host processor 150. Probe 130 functions as an interface between target processor 110 (specifically, TAP 115 of target processor 110) and host processor 150. Probe 130 communicates with target processor 110 via a serial link 160. The operations of probe 130 and host processor 150 for purposes of debugging target processor 110 are generally well known.

[1015] Probe 130 and host processor 150 together emulate an overlay region of memory referred to as a dmseg (*i.e.*, debug memory segment) memory 140 that can be accessed by target processor 110 during debug mode operations. For example, when host processor 150 enters debug mode, a debug exception handler might lie in dmseg memory 140. Hence, the first instruction fetched by host processor 150 would be an address in dmseg memory 140. This dmseg memory 140 is emulated using memory associated with probe 130, memory associated with host processor 150, or some combination of the two. The operation of dmseg memory 140 for purposes of debugging target processor 110 is also generally well known.

[1016] FIG. 2 illustrates various registers included in TAP 115 that are accessed via serial link 160 that operates as a communication channel between target processor 110 and probe 130 when target processor 110 operates in debug mode. These registers include, among others, a Control register 210, an Address register 220, a Data register 230, and an Instruction register 240. Control register 210 includes various status flags that control features and operations associated with the EJTAG Specification. Address register 220 allows target processor 110 to indicate a specific address in dmseg memory 140 that it wishes to access. Data register 230 allows target processor 110 to transfer data to and from dmseg memory 140. Instruction register 240 provides a mechanism for probe 130 and host processor 150 to select one of (or a combination of) the other registers to which to write or from which to read. The operations of these four registers are also generally well known.

[1017] Before describing the present invention in further detail, a description of how conventional EJTAG debug systems transfer large blocks of data or instructions is provided with reference to FIG. 3. For purposes of illustration, the following discussion describes how a block of data is transferred from probe 130 to target processor 110, or more specifically, from dmseg memory 140 to target memory 120. A similar operation is required to transfer a block of data from target memory 120 to dmseg memory 140 as will be apparent from the following description.

[1018] In an operation 310, Control register 210 is selected in order for probe 130 to read the contents of Control register 210, namely to determine whether a processor access pending ("PrAcc") bit is set. When target processor 110 attempts to access dmseg memory 140 while in debug mode, the PrAcc bit in Control register 210 is set. The PrAcc bit functions as a handshake between target processor 110 and probe 130 indicating that target processor 110 is ready to

access dmseg memory 140 with a load, a store, or a fetch operation. In one embodiment of the present invention, Control register 210 is selected by clocking an appropriate 5-bit command from probe 130 into Instruction register 240 of TAP 115. This command prepares TAP 115 to provide probe 130 with the contents of Control register 210.

[1019] In an operation 320, Control register 210 is read to determine whether there is a pending access by target processor 110 of dmseg memory 140. More specifically, the PrAcc bit in Control register 210 is read to determine this pending access. Probe 130 periodically polls Control register 210 to determine whether a processor access is pending by clocking the entire Control register 210 from TAP 115 to probe 130. Once the contents of Control register 210 are received, probe 130 is able to access PrAcc bit to determine whether target processor 110 is waiting to read from or write to dmseg memory 140.

[1020] When a processor access is pending, in an operation 330, Address register 220 is selected so that the address in dmseg memory 140 where the data is to be read can be transferred from target processor 110 to probe 130. In one embodiment of the present invention, this is accomplished by clocking an appropriate 5-bit command from probe 130 into Instruction register 240 of TAP 115. This command prepares TAP 115 to provide the contents of Address register 220 to probe 130.

[1021] In an operation 340, m-bits (dependant upon an address space of dmseg memory 110) of address from Address register 220 are clocked from TAP 115 to probe 130. This is the address in dmseg memory 140 from which data is to be retrieved and placed in Data register 230.

[1022] In an operation 350, Data register 230 is selected so that data can be transferred from probe 130 to target processor 110, or again, more specifically, from dmseg memory 140 to target memory 120. In one embodiment of the present invention, this is accomplished by clocking an

appropriate 5-bit command from probe 130 into Instruction register 240 of TAP 115 to select Data register 230. Once Data register 230 is selected, when operation 360 is a write operation, n-bits of data (dependant upon a word size of target processor 110) are clocked from probe 130 to target processor 110. More specifically, n-bits of data from the address in dmseg memory 140 indicated by Address register 220 are clocked into Data register 230. Target processor 110 can then store the contents of Data register 230 to target memory 120. Store operations are similar as would be apparent from the above description. Specifically, when operation 360 is a read operation (*i.e.*, an upload or transfer of a block of data from target memory 120 to dmseg memory 140), Address register 220 contains the address of dmseg memory 140 where data is to be stored, Data register 230 contains the data to be stored in dmseg memory, and operation 360 clocks the contents of the Data register to probe 130, which then stores the data in dmseg memory 140.

[1023] In an operation 370, after clocking in the data to Data register 230, Control register 210 is again selected, this time to write to its contents. In an operation 380, probe 130 clears PrAcc bit (in one embodiment of the present invention) to indicate that the pending access to dmseg memory 140 by target processor 110 has been satisfied by probe 130. These same operations would also occur after a store operation, as would be apparent from the above description.

[1024] Operations 320-380 transfer a single data word from probe 130 to target processor 110. These operations must be repeated in a conventional EJTAG debug system for each data word transferred to and from dmseg memory 140 as indicated by the loop of FIG. 3. In order to accomplish this transfer, a total of 79+m bits of overhead are required to transfer n bits of data.

This is highly inefficient and time consuming, particularly over a relatively slow serial communication channel.

[1025] Moreover, as noted above, operations 320-380 must be repeated for each word in a block of data that is transferred to and from dmseg memory 140. Such block transfer is achieved, for example, by host processor 150 creating a simple loop routine that causes target processor 110 to carry out multiple load or store operations. As is well known, such a routine may be created by host processor 150 and downloaded to target memory 120 via probe 130 (using the process described in FIG. 3, for example). Target processor 110 is then made to jump to the routine (directed by probe 130) to carry out the block transfer. When the transfer is completed, target processor 110 jumps out of the routine. Because host processor 150 creates the routine, it specifies the addresses that make up the block of data loaded from or stored to target memory 120. Similarly, host processor 150 defines which address or addresses are accessed in dmseg memory 120 to carry out the block transfer. If a single memory location in dmseg memory 140 is repeatedly specified, host processor 150 will ensure that the memory location is timely serviced to provide or retrieve the necessary data when writing to and reading from respectively, Data register 230 (for example).

[1026] Operations 320-380 represent a conservative and safe approach to transferring data or instructions from probe 130 to target processor 110. Some assumptions might be made to eliminate one or more of these operations. For example, when the operational speed of target processor 110 is far greater than the operational speed of probe 130, a developer may assume during block transfers that each time through the loop of operation 300, target processor 110 has completed its other tasks and is again waiting for probe 130 to transfer another data word. More specifically, the determination, in operation 320, of whether a processor access is pending will be

assumed. This assumption eliminates a need to read Control register 210 as performed by operation 320. Another assumption that may be made by the developer is that an address in Address register 220 is known each time through the loop. For example, the developer may pre-establish a particular type of block transfer between probe 130 and target processor 110 such that these types of block transfers will occur at a predetermined address. This assumption eliminates the need to select and read Address register 220 for each data word transferred. Using the two assumptions thus described, the loop of operation 300 may be reduced to operations 350-380. Although this reduced loop runs faster than the previously described loop, it still requires a fair amount of overhead to transfer blocks of data or instructions. Moreover, the two assumptions made above render the loop "unsafe" and make error recovery somewhat difficult.

[1027] To further increase the speed of block transfers between probe 130 and target processor 110 while eliminating the need to make the aforementioned assumptions, TAP 115 of the present invention includes a Fastdata register 250. Fastdata register 250 provides a mechanism whereby operation 300 of the conventional test system is replaced with a single register data transfer operation as will be described below in further detail. In conjunction with Fastdata register 250, various embodiments of the present invention may also include a FASTDATA instruction. The FASTDATA instruction will also be described in further detail below.

[1028] In one embodiment of the present invention, Fastdata register 250 is a one-bit read/write register where the single bit is referred to as a SPrAcc bit. TABLE I summarizes the operation of Fastdata register 250. As reflected in TABLE I, TAP 115 of the present invention implements Fastdata register 250 as follows. A request for a Fastdata access succeeds if 1) a processor access is pending (*i.e.*, PrAcc has been set), and 2) the Fastdata access is to a particular

area of dmseg memory 140. The first requirement ensures that target processor 110 is waiting for probe 130 to satisfy the Fastdata access as opposed to merely assuming that one is pending as described.

TABLE I - Fastdata Register Operations

Fields		Action	Result
Name	**Bits**		
SPrAcc	0	Clearing SPrAcc. In one embodiment, this in accomplished by shifting in a value of '0'.	Requests completion of the Fastdata access. The PrAcc bit in the control register is overwritten with zero when the access succeeds. (The access succeeds if PrAcc is one and the operation address is in the legal dmseg Fastdata area.) A value of '1' for SPrAcc is shifted out when the access succeeds; a value of '0' for SPrAcc is shifted out when the access fails.
		Setting SPrAcc. In one embodiment, this is accomplished by shifting in a value of '1'.	The PrAcc bit in the control register is unchanged. A value of '1' for SPrAcc is shifted out to indicate that the access would have been successful if allowed to complete; a value of '0' for SPrAcc is shifted out to indicate the access would not have been successful if allowed to complete.

[1029] With regard to the second requirement, according to the present invention, only a predetermined portion of dmseg memory 140, referred to as the Fastdata area (*e.g.*, 0xF..F20.0000 - 0xF..F20.000F), is used for Fastdata transfers. Hence, the second requirement ensures (rather than assumes) that the processor access is in fact a Fastdata access of the Fastdata area as opposed to some other address within dmseg memory 140. For example, if target processor 110 gets an exception during a block transfer while in debug mode, target processor

110 will attempt to reenter the debug exception handler. This will change the address of Address register 220 to an area in dmseg memory 140 outside of the Fastdata area. In this situation, the Fastdata access should fail; and according to the present invention, such accesses do in fact fail.

[1030] If either of the requirements described above are not satisfied, in one embodiment of the present invention, a value of '0' is shifted out of Fastdata register 250 indicating that the Fastdata access failed. When both requirements are satisfied, a value of '1' is shifted out of Fastdata register 250 indicating that the Fastdata access succeeded. In one embodiment, the value to be shifted out of Fastdata register 250 is controlled by hardware (*e.g.*, combinatorial logic) that uses the PrAcc value in Control register 210 and the address in Address register 220 as inputs. Of course, such functionality may be implemented in other ways, including software.

[1031] In order for host processor 150 and probe 130 to access the functionality of Fastdata register 250 as described in Table I, a FASTDATA command or instruction is provided. When the FASTDATA instruction is clocked to Instruction register 240 of TAP 115, TAP 115 configures a TDI-to-TDO data path 400 (*i.e.*, a path of serial data from test data in port to test data out port) as illustrated in FIG. 4. As illustrated in FIG. 4, Data register 230 is serially coupled to Fastdata register 250 so that data clocked into a TDI port 410 from probe 130 over serial link 160 is clocked through Data register 230 and into Fastdata register 250, and subsequently clocked out to a TDO port 420. Thus, when using the FASTDATA instruction, an extra bit (*i.e.*, the SPrAcc bit) is clocked to TAP 115 for each data word to be transferred between probe 130 and target processor 110. As illustrated in FIG. 4, the SPrAcc bit is clocked in front of the data word although in other embodiments of the present invention where the order of Data register 230 and Fastdata register 250 are reversed, the SPrAcc bit may be clocked after the data word as would be apparent.

[1032] As indicated in TABLE I, clocking in a value of '0' for SPrAcc bit will attempt to complete the Fastdata access and when successful, will automatically change the PrAcc bit in Control register 210 to a value of '0', thereby eliminating the need for operations 370 and 380 illustrated in FIG. 3. This further speeds the operation of the conventional block transfer loop. In one embodiment, the new value of PrAcc bit is controlled by hardware (*e.g.*, combinatorial logic) that uses the SPrAcc value shifted into Fastdata register 250, the address in Address register 220 and the current value of PrAcc in Control register 210. Of course, such functionality may be implemented in other ways, including software.

[1033] Fastdata register 250 is used to efficiently transfer blocks of data between dmseg memory 140 and target memory 120. As described herein, an "upload" is defined as a sequence of loads from target memory 120 and stores to dmseg memory 140, while a "download" is defined as a sequence of loads from dmseg memory 140 and stores to target memory 120. These sequences are both performed by target processor 110.

[1034] During Fastdata uploads and downloads, target processor 110 is configured to "stall" on accesses to the Fastdata area of dmseg memory 140. When so stalled, the PrAcc bit in Control register 210 will indicate that target processor 110 is waiting for probe 130 to complete the access. Both upload and download accesses are attempted by clearing Fastdata register 250 (*e.g.*, by shifting in a value of '0' for SPrAcc) and thereby requesting access completion. A value of SPrAcc subsequently shifted out of Fastdata register 250 indicates whether the attempt will be successful as described above (*i.e.*, there was a processor access pending and a legal Fastdata area address was used). Downloads will shift in the data to Data register 230 to satisfy the load from dmseg memory 140 and uploads will shift out the data from Data register 230 to satisfy the store to dmseg memory 140.

[1035] According to the present invention, Fastdata register 250 operates in a manner similar to the PrAcc bit of Control register 210. By placing Fastdata register 250 in TDI-to-TDO path 400 with Data register 230, both data and control functions are achieved without having to switch between Data register 230 and Control register 210.

[1036] TABLE II summarizes the operation of one embodiment of the present invention when various values of SPrAcc are shifted into and out of Fastdata register 250 for Fastdata downloads and Fastdata uploads, respectively.

TABLE II - Operation of FASTDATA Accesses

Probe Operation	Address in FastData Area?	PrAcc in the Control Register	Value of SPrAcc Shifted into FastData Register	Action in the Data Register	PrAcc in Control Register Changes to	Value of SPrAcc Shifted out of FastData Register	Data Shifted Out of Data Register
FastData Download	No	x	x	none	unchanged	0	invalid
	Yes	1	1	none	unchanged	1	invalid
		1	0	write data	0 (Same as SPrAcc)	1	valid (previous) data
		0	x	none	unchanged	0	invalid
FastData Upload	No	x	x	none	unchanged	0	invalid
	Yes	1	1	none	unchanged	1	invalid
		1	0	read data	0 (Same as SPrAcc)	1	valid data
		0	x	none	unchanged	0	invalid

[1037] FIG. 5 illustrates an operation 500 of an EJTAG debug system utilizing the present invention while transferring a large block of data between dmseg memory 140 and target memory 120. Specifically, the following discussion describes how a block of data is transferred from dmseg memory 140 to target memory 120. As will become apparent, a similar operation would be required to transfer a block of data from target memory 120 to dmseg memory 140

(*i.e.*, operation 510 would stay the same and operation 520 would be "read data and fastdata from target processor and forward to probe for storage in dmseg memory").

[1038] In an operation 510, Data register 230 and Fastdata register 250 are selected as illustrated in FIG. 4. In one embodiment of the present invention, this is accomplished using the FASTDATA instruction. As described above, the FASTDATA instruction is clocked from probe 130 into Instruction register 240 of TAP 115. The FASTDATA instruction configures Data register 230 and Fastdata register 250 as TDI-to-TDO path 400. In an operation 520, one-bit corresponding to SPrAcc and n-bits of data are clocked from probe 130 to target processor 110. Operation 520 is repeated for each data word in the block of data to be transferred to target memory 120 (via a loop routine running on target processor 110, as described above). Using Fastdata register 250, only one bit of overhead is required (in addition to the initial 5 bits of set up for the FASTDATA instruction) in order to transfer n bits of data and no assumptions are made as to whether a processor access in pending or the address is correct (as described above). This is a significant improvement over the conventional test system described above.

[1039] In particular, this reduction in overhead is achieved with Fastdata register 250 for several reasons. First, probe 130 does not need to separately select and read Control register 210 in order to determine if a processor access is pending via the PrAcc bit. TAP 115 implements Fastdata register 250 so that the status of PrAcc is automatically determined (such as by using hardware) thereby preventing the Fastdata access from completing if a processor access is not pending. Second, probe 130 does not need to select and read Address register 220 to confirm it is accessing the proper address. While using the Fastdata instruction, Address register 220 need not be accessed because TAP 115 implements Fastdata register 250 so that only an access to the Fastdata area of dmseg memory 140 will be allowed to complete. This may be achieved, for

example, by using hardware that automatically confirms that the current address falls within dmseg memory 140. Third, probe 130 does not need to separately select and write Control register 210 in order to indicate that the pending access has been satisfied. According to the present invention, after the Fastdata access successfully completes, the PrAcc bit is automatically reset.

[1040] While the invention has been described in detail and with reference to specific embodiments thereof, it will be apparent to one skilled in the art that various changes and modifications can be made therein without departing from the spirit and scope thereof. For example, the operation of the present invention is described in terms of shifting a certain value into or out of Fastdata register 250. As would be apparent, various other values may be used to implement the same or similar functionality. Similarly, any other mechanism for retaining and shifting data may be used in place of Fastdata register 250.

[1041] Moreover, in addition to implementations of TAP 115 embodied in hardware (*e.g.*, within a microprocessor, microcontroller, SOC component, etc.), implementations may also be embodied in software disposed, for example, in a computer usable (*e.g.*, readable) medium configured to store the software (*i.e.*, computer readable program code). The program code causes the enablement of the functions or fabrication or both of the TAP-related structure and functionality described herein. For example, this can be accomplished through the use of general programming languages (e.g., C, C++), hardware description languages (HDL) including Verilog HDL, VHDL, and so on, or other available programming and/or circuit (e.g., schematic) capture tools. The program can be disposed in any known computer usable medium including semiconductor, magnetic disk, optical disk (e.g., CD-ROM, DVD-ROM) and as a computer data signal embodied in a computer usable (e.g., readable) transmission medium (e.g., carrier waves

or any other medium including digital, optical or analog-based medium). As such, the code can be transmitted over communication networks including the Internet and intranets.

[1042] It is understood that the TAP-related functions and/or structures provided above can be represented in a core (e.g., microprocessor core), SOC component, etc., that is embodied in program code and may be transformed to hardware as part of the production of integrated circuits. Also, these functions and/or structures may be embodied in a combination of hardware and software. Thus, it is intended that the present invention cover the modifications and variations of this invention provided they come within the scope of the appended claims and their equivalents.